Institute of Informatics, Warsaw University

Behavioural specifications

Władysław M. Turski
wmt@mimuw.edu.pl

TR 95-02(202)
January, 1995

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	January 1995	

4. TITLE AND SUBTITLE ~~A Dynamical Model of~~ Behavioural Specifications

5. FUNDING NUMBERS
F6170894C0001

5. AUTHOR(S)
Turski, W.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Institute of Informatics
Banacha 2
Warsaw 02-097
Poland

8. PERFORMING ORGANIZATION REPORT NUMBER
TR 95-02 (202)

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

EOARD
PSC 802 Box 14
FPO 09499-0200

10. SPONSORING / MONITORING AGENCY REPORT NUMBER
SPC-94-4005-4

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. *ABSTRACT* (Maximum 200 Words)

14. SUBJECT TERMS Foreign Reports, EOARD

NUMBER OF PAGES
26

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

Institute of Informatics
Warsaw Univerity
ul. Banacha 2
02-097 Warsaw, Poland
phone: (48-2) 658-31-65
fax: (48-2) 658-31-64

Druk: Zaklad Graficzny UW zam.nr.

Contents

Behavioural specifications

Władysław M. Turski

0 Motivation

For historic reasons, programming (and its theory, as well as methodology) evolved from the *computing paradigm*. Many computer applications in common use do not fit this paradigm well. Neither an operating system, nor a word processor *compute* anything, even if their operation involves many computations.

Here we follow another paradigm, *viz.* that of *system behaviour*, in which finite *actions* are undertaken and accepted in specific circumstances. In this approach there is no notion of action sequencing, therefore we avoid all problems of synchronization and global time. Instead, we recognize that action executions are not instantaneous and—therefore—the circumstances under which a particular action was deemed desirable may have changed during an execution to ones under which its outcome is no more needed. (Establishing a telephone connection in a modern network is an action that certainly takes some time during which the caller may decide to hang up; should this be the case, the labouriously established connection may safely be dissolved.)

The chosen paradigm accommodates an arbitrary degree of parallelism in the sense that an unspecified number of actions may be executed concurrently and—as far as the execution processes are concerned—independently. All actions that *can* be initiated in a given state, *are* initiated, regardless of any possible future conflict; conflicts are resolved in the accepting state. (When a client asks the bank to cash a cheque, the teller starts counting the money and checks the balance only when handing the cash out.)

Observable system behaviour obtains from successful executions of actions. For each system the collection of actions is determined by its specification that associates each action with *two* guards. A *preguard* characterizes the set of system *states* in which an action may be undertaken, a *postguard* characterizes the set of states in which the outcome of the executed action is acceptable *i.e.* may be reflected in the system state. Each action is assumed finite if undertaken in a state satisfying the corresponding preguard. Actions are performed by *agents* of which there is an unspecified number (often it is essential that the number of agents exceeds one). Each agent is capable of executing any action, but different agents may need different lengths of time to complete the same action. No agent is allowed to idle in a system state in which a preguard is satisfied. Actions are assigned to agents in a fair way: when more than one can be assigned, the actual choice is unbiased.

If, at the instant of action completion the system state[1] satisfies the corresponding postguard, the action outcome is instantaneously accepted.

In other words, the paradigm aims to capture behaviour exhibited by systems in which actions take time (and may be futile) but all controls and state-updates are instantaneous.

[1] Note that had we taken not the *system state*, but the *agent's private state* as the domain on which to evaluate the acceptance criterion, we would have obtained the model proposed by Randell [5] for programming reliable systems.

1 Basic notions and constructs

States of the system are represented by (vector) values of a variable z, possibly with subscripts. Other lower case italic bold-face letters denote projections of the system state on subsets[2] of state coordinates. Individual coordinates (state variables) will be denoted by italic letters, possibly with subscripts. Thus, when $\boldsymbol{z} = (z_1, z_2, z_3, z_4, z_5, z_6)$, projection on state variables (z_2, z_4) could be denoted, for example, by $\boldsymbol{x}$, and (z_1, z_2, z_5, z_6) by $\boldsymbol{y}$.

Truth valued functions of the state, known as *predicates*, will be denoted by capital italic letters $P, Q, R, \ldots$, possibly with subscripts. Function application will be denoted by the dot, thus *e.g.* $P_i.\boldsymbol{z}_0$ denotes application of P_i to $\boldsymbol{z}_0$. When the application of a predicate to a state yields **true** we say that this state *satisfies* the predicate.

Actions will be denoted by $a, b, \ldots$, possibly with subscripts. Actions may be identified with their programs.

$$(P, Q) \rightarrow [\boldsymbol{x}|a|\boldsymbol{y}]$$

is the notation employed for specification of an action a that may be undertaken in a state satisfying predicate P, and whose outcome is accepted iff at the instant of its termination the system state satisfies predicate Q. Variables in $\boldsymbol{x}$ constitute the inputs to a, variables in $\boldsymbol{y}$, its output. Actions are executed on a *private* state, which includes all variables in $\boldsymbol{x}$ and in $\boldsymbol{y}$. Whenever the inputs and/or outputs to a are readily identified by the context, *e.g.* constitute the whole vector state, the corresponding parts of the notation may be skipped. Thus occasionally we may write $(P, Q) \rightarrow [a]$, $(P, Q) \rightarrow [\boldsymbol{x}|a]$ or $(P, Q) \rightarrow [a|\boldsymbol{y}]$.

It is assumed that $P \Rightarrow wp.(a, \mathbf{true})$, *i.e.* a properly initialized action terminates.

Let $\boldsymbol{z}_b$ be a state satisfying P and let action a be selected for execution in it. Variables of $\boldsymbol{x}$ in the private state of a instantaneously obtain values equal to those of corresponding variables in $\boldsymbol{z}_b$ and the execution of a starts. When it terminates, the system is in state $\boldsymbol{z}_m$. If $\boldsymbol{z}_m$ satsifies Q, it is instantaneously modified by the results of action a, which effectively turns it into another state, $\boldsymbol{z}_e$, identical to $\boldsymbol{z}_m$ except for variables in $\boldsymbol{y}$ which in $\boldsymbol{z}_e$ have values produced by the execution of a. On the other hand, if $\boldsymbol{z}_m$ does not satisfy Q, the results of action a are totally ignored. In either case the agent which performed a is released and becomes available for another task.

If, for all $\boldsymbol{z}_b$ satisfying P and for a particular $\boldsymbol{z}_m$ satisfying Q, upon acceptance of a, obtains $\boldsymbol{z}_e \neq \boldsymbol{z}_m$, we say that action a is *productive in* $\boldsymbol{z}_m$. If $Q.\boldsymbol{z}_m$ implies that a is productive in $\boldsymbol{z}_m$ we say that a is *productive*. (The requirement that actions be productive is not very restrictive from a practical point of view: it hardly makes sense to specify an action whose outcome would not change the very state in which it is acceptable.)

Consider some interesting specifications:

- $(P, \neg P) \rightarrow [\boldsymbol{x}|a|\boldsymbol{y}]$ specifies an action whose results may be accepted iff during its execution the system state was changed from one satisfying P to one satisfying $\neg P$. Whatever the actual meaning of the predicate P, this implies that the system state has changed. Since it was not (yet!) changed by the execution of action a, the change—if it occurred—is due to some other action, obviously executed by another agent. For a *closed* system (*i.e.* one whose state changes *only* by the execution of explicitly specified actions) this means that the specified action cannot be successfully realized by a single agent. This must be the shortest specification of a computation which cannot be executed by a single processor!

[2] In this report, the subsets of variables are selected mostly for their rôle, *e.g.* as input or output variables.

- $(P, P) \rightarrow [\boldsymbol{x}|a|\boldsymbol{y}]$ specifies an action whose results may be accepted iff during its execution the system state remains unchanged or is restored to one satisfying the same predicate as the state in which it was undertaken.
- $(P, \mathbf{true}) \rightarrow [\boldsymbol{x}|a|\boldsymbol{y}]$ specifies an action whose results are always acceptable (because **true** is a predicate satisfied by all states). It is recommended that actions that modify the "external world", such as physical output or manipulation of actuators in control systems, be specified in this manner.
- $(P, \mathbf{false}) \rightarrow [\boldsymbol{x}|a|\boldsymbol{y}]$ specifies an action whose results are never acceptable. (It is hard to imagine any use of such a specification.)
- $(\mathbf{true}, Q) \rightarrow [\boldsymbol{x}|a|\boldsymbol{y}]$ specifies an action that may be undertaken in any state. This may be employed to specify a "busy wait" delivering useful results if only Q is satisfied at any time they are ready.
- $(\mathbf{false}, Q) \rightarrow [\boldsymbol{x}|a|\boldsymbol{y}]$ specifies an action that can never be undertaken. (A totally useless construct; its emergence, *e.g.* ensuing from a simplification of a specification, signifies a major design error of the original specification.)

Primary objects of interest in this report are specifications of the form

$$(P_i, Q_i) \rightarrow [\boldsymbol{x}_i|a_i|\boldsymbol{y}_i] \text{ for } 0 \leq i < N \tag{1}$$

which are intended to prescribe total system behaviour. For the most part we are concerned with closed systems; whenever this constraint is relaxed, we say so explicitly.
Note. When it is certain that there is but one processor available to execute the implementation of (1), the specification is equivalent to

$$(P_i \wedge Q_i, \mathbf{true}) \rightarrow [\boldsymbol{x}_i|a_i|\boldsymbol{y}_i] \text{ for } 0 \leq i < N$$

or, in a more familiar form of Dijkstra's guarded statements[3] (*cf.* [2]), to

$$\mathbf{do} \, [] \, P_i \wedge Q_i \Longrightarrow a_i \, \mathbf{od}$$

Indeed, while the single available processor is executing a_i, in a closed system, there is no other agent capable of establishing Q_i. Thus if it is expected that Q_i holds when the execution of a_i terminates, it must be the case that Q_i was satisfied at the initialisation of a_i. This observation establishes a link between semantics of (1) and the well-known semantics of guarded statements to which it reduces in the case of a single processor system. *End of Note.*

If—for any reason—we wish to define the *initial state*, specification (1) may be augmented by the *initialization statement, e.g.*

$$\begin{aligned} \{\boldsymbol{z} \;&:=\; \ldots\} \\ (P_i, Q_i) \;&\rightarrow\; [\boldsymbol{x}_i|a_i|\boldsymbol{y}_i] \text{ for } 0 \leq i < N \end{aligned}$$

where the initial state obtains by execution of the multiple assignment. The braces are added as a reminder to indicate that the initialization statement is executed only once.

[3] In order to differentiate between guarded actions and Dijkstra's guarded statements, in the latter we employ double arrows $\Longrightarrow$.

Actually, much more important than the setting of variable values is the fact that in the initial state there are no "pending" actions, *i.e.* actions initiated before and not yet completed (*cf.* Section (3), particularly the footnote to p. 9). In any other state, besides actions that may be initiated because the state satisfies their preguards, we need to be aware of pending actions, in particular, of such pending actions may be acceptable in this state. An action initiated in state z need not be acceptable in this state, in which case a move from z can be effected only by acceptance of a pending action.

If in a given state more than one action can be initiated, the actual choice is purely random (fair). If more than one agent is available, several actions or even several copies of the same action may be initiated. This is particularly likely for an initial state in which—initially—all agents are available and not until first of the assigned actions is successfully completed and accepted can the conditions change.

On the other hand, if in a given state z only one action, a, can be initiated, this action is initiated as soon as the state is established. Indeed, the state z is established by accepting the results of a completed action, which is accompanied by the release of an agent. Agents not being allowed to idle in the presence of satisfied preguards, the released agent is assigned to action a (the only one with its preguard satisfied in z).

Thus—as long as there is no choice to be made—actions that can be initiated are initiated without any delay. This observation is independent of the (positive) number of agents available and of the speed of actions execution. The fan-out degree (the number of concurrently initiated actions) is stochastically dependent on these "implementation details".

Since no information on actual execution times for various action/agent assignments is assumed to be available, we cannot calculate the state as a function of time. Since, however, it is assumed that all actions undertaken in states satisfying their preguards are finitely executable by any agent, and any action execution takes a positive length of time, in any finite interval of time any closed system will be in a finite number of different states only.

Consider a finite interval $[t_0, t_1]$. The *set* of states exhibited by the system in this interval, denoted by $O[t_0, t_1]$, will be referred to as $[t_0, t_1]$-segment of the system *orbit*. With our lack of information on implementation details, $O[t_0, t_1]$ is not very informative. Indeed, in different implementations $O[t_0, t_1]$ may consist of different states; even different runs of one implementation may produce different $O[t_0, t_1]$.

Let us assume, however, that we observe state z_0 and measure its epoch (time of occurrence) t_0 (or, alternatively, that we observe the system at time t_0 and mark its state z_0). We say that z_0 is a *stationary state* iff all possible orbit segments $O[t_0, t_1]$ for arbitrary $t_1 > t_0$ are singletons $\{z_0\}$. Note that this definition has one free variable only, *viz.* t_0, representing z_0; once the state in question is fixed, the definition is implementation-independent.

In general, we are not interested in system *trajectories, i.e.* time-ordered *sequences* of states as these are even more implementation-dependent than orbit segments. If, however, we are given a system trajectory, or its segment $z_0, z_1, \ldots, z_n$, then, given $0 \leq i < n$ $(0 < j \leq n)$, states $z_{i+1}, \ldots, z_n$ $(z_0, \ldots, z_{j-1})$ are known as successors (predecessors) of state z_i, and state z_{i+1} (z_{j-1}) is known as its direct successor (predecessor). In general, neither successors, nor direct successor is uniquely determined. As to the predecessors, although they cannot be uniquely inferred from a state, we accept that each actual state has a unique history; the past behaviour of an implemented system is fixed and may have been recorded.

2 Properties of states

Let S be a predicate satisfied by stationary states and only by these states:

$$S.z \equiv \text{state } z \text{ is stationary}$$

Consider now a behavioural specification (1) and let $\alpha \stackrel{\text{def}}{=} \{i : 0 \leq i < N\}$. For a given z we may define two useful sets of integers: $\beta \stackrel{\text{def}}{=} \{i : i \in \alpha \wedge Q_i.z\}$ and $\gamma \stackrel{\text{def}}{=} \{i : i \in \alpha \wedge P_i.z\}$. Introducing a set of vectors from which the system states are drawn, Z, by the usual abuse of notation we can employ β and γ as the names of functions of the type $Z \rightarrow 2^{\mathcal{I}}$, in which context α is a set-valued constant. Thus $\beta.z$ ($\gamma.z$) is the set of subscripts of postguards (preguards) satisfied in z.

Property 1

$$\beta.z = \emptyset \Rightarrow S.z$$

Proof. If no action results may be accepted in state z ($\beta.z = \emptyset$) then there is no way for this state to ever change, all orbit segments starting with the instant in which state z is established are singletons $\{z\}$. QED.

Note. A state z may be stationary without necessarily $\beta.z = \emptyset$; indeed, the state may be receptive for results of some actions, but if these were not initiated in the past, no potential state changes will occur if the actions that can be initiated in z are not acceptable in it. *End of Note.*

Property 1′

$$\neg S.z \Rightarrow \beta.z \neq \emptyset$$

Proof. By elementary calculus. QED.

Property 2 Let $\delta.z \stackrel{\text{def}}{=} \beta.z \cap \gamma.z$. If all actions a_i for $i \in \beta.z$ are productive

$$\delta.z \neq \emptyset \Rightarrow \neg S.z$$

Proof.

$$\begin{aligned} & \delta.z \neq \emptyset \\ \equiv\ & \beta.z \cap \gamma.z \neq \emptyset \\ \equiv\ & \{i : i \in \alpha \wedge Q_i.z\} \cap \{i : i \in \alpha \wedge P_i.z\} \neq \emptyset \\ \Rightarrow\ & (\exists j : j \in \alpha : Q_j.z \wedge P_j.z) \end{aligned}$$

thus there exists at least one action, a_j, which can be initiated and accepted in z. If a pending action is accepted before action a_j is completed, z is changed to $z' \neq z$ (all acceptable actions are assumed productive) and orbit segments will contain at least two states. If no pending action is accepted before a_j is completed, the state will change upon acceptance of a_j results (being acceptable, a_j is also productive). QED.

Note. Strictly speaking, it would be enough to assume that the acceptable actions are productive in z. *End of Note.*

Property 2′ If all actions a_i for $i \in \beta.z$ are productive

$$S.z \Rightarrow \delta.z = \emptyset$$

Proof. By elementary calculus. QED.

Property 3 A sufficient condition for the state z to possess a unique direct successor is $(\forall i : i \in \delta.z : z' \neq z \Rightarrow \neg P_i.z') \wedge \text{card}.(\delta.z) = 1$.

Proof. The first conjunct of the condition ensures that only actions initiated in z may be accepted in this state, the second one, that there is exactly one action that can be both initiated and accepted in z; upon acceptance, this action establishes the (unique) successor to z. QED.

Note 1. It is immaterial whether the accepted action is freshly initiated or pending (initiated at some past time when the system was also in z and slowly executed): the results of any action are fully determined by its program and by the initial state which in either case is the same. *End of Note 1.*

Note 2. A temptingly simpler condition, $\text{card}.\beta.z = 1$, fails to guarantee the existence of a successor. *End of Note 2.*

Corollary to Property 3 For an initial state the sufficient condition for having a unique successor reduces to $\text{card}.(\delta.z) = 1$.

Proof. Initial states have no pending actions. QED.

3 An alternative view

In Section 2 an important rôle was played by the function $\delta.z$. If only one agent is executing the specification (1), and thus there are no pending actions in *any* state, the future behaviour of the closed system observed in a state z is completely determined by actions whose indices are in $\delta.z$. (This is not to say that the future is necessarily deterministic, nor that the state z has a unique successor!) The reasoning about behaviour is correspondingly simplified, *e.g.*, we can get rid of the awkward first conjunct in the sufficient condition for unique successor (Property 3): the simpler version occurring in the Corollary to Property 3 applies to all states.

If we have many concurrently acting processors, the pending actions have to be reckoned with in all states except the initial one, the "menace" of a slow-acting processor executing an action initiated long ago and attempting to modify the current state (if only the postguards allow it) is always present.

In this context it is worthwhile to consider the set $B.z \stackrel{\text{def}}{=} \{a_i : i \in \beta.z\}$. The state z can be modified only by an action from $B.z$, thus if the designer of a system is worried whether a particular state can or cannot be modified by a particular action, the only thing she has to do is to compute $B.z$; note that this is a fully "static" procedure, requiring no knowledge of system orbit, let alone its trajectory.

The set $B.z$ is (usually) "too large" in the sense that it includes actions that need not have been initiated in the past, as well as actions initiated and completed before state z was established. Even if an action from $B.z$ has been initiated and has not been completed yet, there is no guarantee it will not be preempted by another one, which—being productive in z—will change the state, and with it, perhaps, the acceptability criteria. Thus the set $B.z$ is much more useful for affirming that a specific action *will* not modify a given state. This is often of practical significance ("no valve can close in this state").

On the other hand, if a historical record of system behaviour is available in the form of the sequence of its states: $z_0, z_1, ..., z_n$, with z_n denoting the current state, the information provided by $B.z_n$ can be refined as follows:

Let $G.z_j \stackrel{\text{def}}{=} \{a_i : i \in \gamma.z_j\}$, $j = 0, ..., n$, and let $\overleftarrow{z_n} \stackrel{\text{def}}{=} z_0, z_1, ..., z_n$. Compute $I.\overleftarrow{z_n} = \bigcup_{j=0}^{n} G.z_j$ and $A.\overleftarrow{z_n} = I.\overleftarrow{z_n} \cap B.z_n$. The set $A.\overleftarrow{z_n}$ refines $B.z_n$ as it does not contain any action acceptable in z_n that

could not have been initiated in the past[4]. This refinement is still too crude as there is no guarantee that an action that could have been initiated was indeed started, nor that an initiated action is still pending, nor—of course—that even if it is pending it will not be preempted.
Note. It is not necessary to actually observe the system behaviour to construct a sequence $z_0, z_1, ..., z_n$, it may represent a hypothetical pattern, about which we ask the question: "*assume* the system runs through this sequence of states, which actions can/cannot happen now?". *End of Note.*

In the above reasoning we have repeatedly formed sets of actions without worrying too much about semantic implications of such acts. As long as we consider the resulting sets merely as unordered lists of *action names* with no repetitions, we are OK, but we should remember that we (silently!) accepted to disregard any possible differences between actions initiated in different states: if we want to distinguish between $[x|\ y := x^2\ |y]$ initiated when $x = 1$, and the same (?!) action initiated when $x = 100$, we must not follow this path.

Two assumptions may lead to an interesting development:

1. There are enough processors to carry out all actions that can be initiated. In its stronger form we assume that any action that can be initiated is started many times on many (perhaps different) processors.

2. All actions are partial constant functions. The preguards ensure that actions can be initiated only within the domains of appropriate functions, but wherever an action is initiated it yields the same result (the postguard determines if this result is accepted).

With these assumptions, in any state we may expect pending a copy of any action that could have been previously initiated; the set $A.\ \overline{z_n}$ represents now all that can happen to state z_n. Indeed, assumption (1) ensures that even if a copy of action was "consumed" in a previous state there are other copies being performed (perhaps on slower processors); assumption (2) ensures that no matter which copy happens to be completed when the system is in state z_n, if only it is acceptable, the modification to z_n will be the same.

Thus, given a history (real or hypothetical) and a bit of static computations, we know all possible changes to z_n. The problem is how to predict which of the possibilities will materialize. This is, of course, a stochastic issue. We hope to address it in the future.

4 Terminating behaviours

In this section we present a theorem that sets a sufficient condition for an autonomous system specified by (1) to terminate its behaviour in utmost a finite number of steps. In other words, we are after a condition that guarantees that all system's trajectories are finite. The development closely follows Dijkstra's theorem on terminating computations (*cf.* [2]), which, as pointed out in [6], is an analogue of Liapunov's theorem on stability (*cf.* [3]).

Assume that the subset of state variables occurring simultaneously in outputs of *all* actions is not empty, i.e. that there are some state variables manipulated by all actions. Let $\emptyset \neq z_y \stackrel{\text{def}}{=} \bigcap_{i=0}^{N-1} y_i$, $0 \leq i < N$, be this subset.

[4]Note that $A.\ \overline{z_n}$ depends on the history (actual or hypothetical) of system behaviour up to and including z_n. If the current state could have been reached via different sequences of states, future behaviours may correspondingly differ. The system is blatantly non-Markovian.

If there exists a function $\lambda : z_y \to \mathcal{N}$ such that

$$(P_i \wedge \lambda.z_y \leq \Lambda + 1) \Rightarrow wp(a_i, \lambda.z_y \leq \Lambda),\ 0 \leq i < N \tag{2}$$

for all natural Λ, then, as long as the number of executing agents remains finite, all autonomous behaviours of (1) terminate. (In (2) *wp* stands, of course, for Dijkstra's *weakest precondition.*)

Proof. Premise (2) guarantees that the execution of any action of (1) establishes a z_y–state at which values of λ are bound by a nonnegative integer one less than the similar bound at the action inception. By the definition of z_y and assumption of its nonemptiness, if this action's results are accepted, in the established *system* state the values of λ are bound by an integer smaller than in the system state in which the action was started.

Let the bound on λ in the initial system state be $\Lambda^0 + 1$. When the last action initiated in this state terminates and is accepted the bound is *decreased* to Λ^0. Note that in view of there being utmost a finite number of agents the notion of "last" is meaningful. Of course, any action terminating and accepted before the last one could have established such decreased bound earlier, but in view of the finite duration of all actions, no trajectory started in the initial state can have had more than a finite number of steps. Since the number of executing agents is also finite, we note that after no more than a finite number of *system* steps the bound on λ is irrevocably decreased by at least one.

The reasoning may be now repeated: actions pending in the current state, when accepted, may only decrease the bound; when the last action initiated in the current step is completed and accepted, a decrease by one is again irrevocable.

Clearly, this cannot continue forever, as the values of λ are—by assumption—nonnegative and in no more than in $\Lambda^0 + 1$ "last action" steps the system achieves a state in which the bound on λ is 0, i.e. (2) would be violated by any action. In this state, by the law of excluded miracles (*cf.* [2]) no P_i can be satisfied, and thus no action can be initiated. Note also that in this state there are no pending actions, as all "faster" branches have exhausted the bound limit sooner. QED.

The general assumption of this section may seem very strong; in many practical cases, however, it is a very natural one. Indeed, consider a specification for a system behaviour in which each action consumes a finite amount of an autonomously non-renewable resource (for instance, fuel is consumed). The z_y set may be simply the variable representing the measure of the resource, and function λ an integer approximation to this measure (*e.g.* fuel gauge readings calibrated in units of the minimum fuel consumption, a well-defined notion in view of the finiteness of the specification).

5 Manipulating the specification

Directly from the definition of guarded actions one can obtain some useful rules to manipulate a specification while preserving its meaning. Consider a specification that includes the following pair of guarded actions

$$\begin{array}{lcl} (P_1, Q_1) & \to & [\boldsymbol{x}|a|\boldsymbol{y}] \\ (P_2, Q_2) & \to & [\boldsymbol{x}|a|\boldsymbol{y}] \end{array} \tag{3}$$

1. If $P_1 \equiv P_2$ then (3) is equivalent to a single guarded action

$$(P_1, Q_1 \vee Q_2) \to [\boldsymbol{x}|a|\boldsymbol{y}]$$

2. If $Q_1 \equiv Q_2$ then (3) is equivalent to a single guarded action

$$(P_1 \vee P_2, Q_1) \to [\boldsymbol{x}|a|\boldsymbol{y}]$$

3. If $P_1 \Rightarrow P_2$ then (3) is equivalent to the pair of guarded actions

$$\begin{array}{rcl}(P_1, Q_1 \vee Q_2) & \rightarrow & [x|a|y] \\ (\neg P_1 \wedge P_2, Q_2) & \rightarrow & [x|a|y]\end{array}$$

4. If $Q_1 \Rightarrow Q_2$ then (3) is equivalent to the pair of guarded actions

$$\begin{array}{rcl}(P_1 \vee P_2, Q_1) & \rightarrow & [x|a|y] \\ (P_2, \neg Q_1 \wedge Q_2) & \rightarrow & [x|a|y]\end{array}$$

Rules 3 and 4 are particularly useful together with 5 or 6.

5. Guarded action

$$(P, \mathbf{false}) \rightarrow [x|a|y]$$

may be eliminated from any specification.

6. Guarded action

$$(\mathbf{false}, Q) \rightarrow [x|a|y]$$

may be eliminated from any specification.

Selection, (and, therefore, useful nondeterminism) may be traded between action bodies and specification structure:

7. The pair of guarded actions

$$\begin{array}{rcl}(P_1, Q) & \rightarrow & [x|a|y] \\ (P_2, Q) & \rightarrow & [x|b|y]\end{array}$$

is equivalent to a single guarded action

$$(P_1 \vee P_2, Q) \rightarrow [x|\ \mathbf{if}\ P_1\ \mathbf{then}\ a\ \mathbf{else}\ b\ \mathbf{fi}\ |y]$$

or, admitting guarded statements in bodies of actions, to

$$(P_1 \vee P_2, Q) \rightarrow [x|\ \mathbf{if}\ P_1 \implies a\ [\!]\ P_2 \implies b\ \mathbf{fi}\ |y]$$

Rule 7 can be extended to actions with different lists of input variables. Indeed, let $x_1 \oplus x_2$ stand for the set-theoretic sum of x_1 and x_2, then

8. The pair of guarded actions

$$\begin{array}{rcl}(P_1, Q) & \rightarrow & [x_1|a|y] \\ (P_2, Q) & \rightarrow & [x_2|b|y]\end{array}$$

is equivalent to a single guarded action

$$(P_1 \vee P_2, Q) \rightarrow [x_1 \oplus x_2|\ \mathbf{if} P_1 \implies a\ [\!]\ P_2 \implies b\ \mathbf{fi}\ |y]$$

Note. A similar trick does not always work for the output variables as each of the two original guarded actions may modify different variables in the accepting state; modifying their set-theoretic sum could lead to a state not foreseen in the specification. *End of Note.*

6 A simple example

Consider a system consisting of a flip-flop and a counter. The flip-flop spontaneously changes its value from 0 to 1 and *vice versa*. Between the changes, which are instantaneous, the flip-flop value remains stable for a finite length of time. The counter records the number of such changes. We shall specify the counter only, thus we are concerned with an *open* system as values of the flip-flop are not determined by actions of the counter. Denoting flip-flop by x, and counter by n, we have the first specification

$$\begin{array}{lcl}(x=0, x=1) & \rightarrow & [n|\, n := n+1|n] \\ (x=1, x=0) & \rightarrow & [n|\, n := n+1|n]\end{array} \tag{4}$$

or, if we wish to set the counter initially to 0, a (partial) initialization statement

$$\{n := 0\}$$

may be added to the specification.

Note. Adding a complete initialization statement $\{x, n := \ldots, 0\}$ would be awkward and somewhat confusing as we do not intend to specify the flip-flop behaviour. *End of Note.*

Since for all possible states of the system specified by (4) we have both $\beta.z \neq \emptyset$ and $\beta.z \cap \gamma.z = \emptyset$, it would be perfectly consistent with Properties 1 and 2 for any state to be stationary if (4) was considered as a closed system. In fact, if only the flip-flop always changes its value after a finite time, no state of the counter is stationary. Thus—on the first glance—the design looks all right; a closer inspection, however, reveals two interesting defects.

1. Assume first that there is only one agent available to execute (4). Let the average time needed by this agent to execute $n := n+1$ be T, and let the average frequency of the flip-flop be ν. Then, if $\nu T \gg 1$, any odd number of changes could be recorded as a single change, while any even number of changes could be recorded as no change. This defect, however real, is quite unavoidable: no discrete[5] counter can faithfully register arbitrarily frequent phenomena, unless it also controls them to the extent of being able to delay their occurrence (as semaphores and other mutual-exclusion schemes imply).

2. If more than one agent is available to execute (4) then we are faced with a possibility of a *time-warp*. An agent assigned to perform $n := n+1$ in a state $(x = 0, n = n_1)$, having spent a long time on execution of the assignment, may complete it in a system state $(x = 1, n = n_2)$, $n_2 > n_1 + 1$. Since the postguard $(x = 1)$ is satisfied in this state, the outcome of the action is accepted and the (global) value of n is set to $n_1 + 1$ (*i.e.* to the value obtained by the slow agent).

The first defect does not admit any general solution (see, however, discussion on implementation at the end of present section); the only way we can avoid it in practice is by means of carefully planned experiments (tests). The second defect can be remedied by an improved design. If, however, it can be assumed (or otherwise guaranteed) that the maximal time required by any agent to complete $n := n + 1 < 2 \times$ minimal switch-time of the considered flip-flop, none of the above defects apply.

A correct design should reflect the principle that upon the acceptance of an action outcome the counter can only increase. To achieve it we introduce another integer-valued variable, m, serving as a

[5] "Discrete" ("digital") in this context means any device ("metal or mental") characterized by a finite switching time. The general observation made here is a close analogue of Heisenberg's Principle.

temporary (provisional) counter:

$$\begin{array}{lrcl} & \{m,n & := & 0,0\} \\ * & (m=n\wedge x=0,\ m=n\wedge x=1) & \rightarrow & [m|\, m:=m+1|m] \\ \star & (m=n\wedge x=1,\ m=n\wedge x=0) & \rightarrow & [m|\, m:=m+1|m] \\ \dagger & (m>n,\ m>n) & \rightarrow & [m|\, n:=m\,|n] \end{array} \tag{5}$$

In this design, the counter n is set only by the action marked with $\dagger$. Provisional counter m can be increased by either of the $*$, $\star$ actions, but—of course—only one of them may be initiated in any particular state of the flip-flop. (Note that in the presence of many agents, several copies of $*$ or $\star$ may be launched in any particular state.) Actions $*$ and $\star$ cannot be initiated if $m > n$, thus all available agents (including the one whose completed action—increase of m—has caused this situation) are forced to select action $\dagger$, whose preguard is satisfied. When the first ("fastest") agent completes $\dagger$-action, and thus updates the counter, it is assigned to this of the $*$, $\star$ actions that is selected by the current state of the flip-flop. (Note that the other copies of the $\dagger$-action will not be accepted as long as $m = n$, established by successful completion of the first copy, holds.)

The extra delay, when actions $*$, $\star$ are inhibited (waiting for the counter n to be set), should be considered as a part of the counter switching time. If this leads to missing some flip-flop changes—a fact that may be established only by an experiment involving a particular implementation of (5) and a particular flip-flop—there is nothing that can be done, short of buying faster hardware for the implementation of (5).

A slow agent executing $*$ or $\star$ will not be able to upset the counter by setting a time-warped value: the preguard in $\dagger$-action makes it into a ratchet. What will happen, however, when the slow agent gets to execute action $\dagger$ itself? Well, it *can* succeed in setting n to an obsolete value of m. Is it very dangerous? Theoretically, yes, because for some length of time the value of the counter may be well below the actual count of flip-flop changes. In practice, the danger is mitigated by two circumstances:

1. The design (5) does not include any *use* of the counter value. It is not beyond the limits of sound design methodology to envisage that the use part of some general design will simply disregard any decreased readings of the counter if it ever gets them, *cf.* the use of *compatible* in Section 9. (And if a time-warped n is not perceived as a step-down in a sequence of readings, then the sampling by the use part is so infrequent that the lower accuracy of reading the counter should be acceptable.)

2. At the instant when a slow agent is registering its (time-warped) value of n there is another agent executing another copy of the $\dagger$-action *initiated with a larger value of* m. Indeed, assume that the state in which the action of the slow agent was initiated had $m = m_1 > n = n_1$. For the agent to earn the "slow" attribute, it must have been overtaken by at least one other agent, who succeeded in establishing $n = m_1$. This event enabled at least one of the $*$, $\star$ actions, whose successful completion established $m = m_2 > m_1$. The agent, who established this value of m was necessarily assigned to action $\dagger$ with $m = m_2 > m_1$. QED. If there were many cycles of faster agents increasing m and n, the temporarily spoiled value of n will be worse, but since the slow agent does not change m, the first of the concurrently running faster agents will find the postguard $m > n$ satisfied and set n to the correct value.

Note. The reasoning we relied upon in point 2 above is far from formal. At the present time we have no suitable calculus yet to present a fully satisfactory formal exposition. Nevertheless, we believe in the engineering soundness of our approach. *End of Note.*

The ability of the design (5) to cope quickly with time-warped values of the counter seems to outweigh the advantages of another design:

$$\begin{array}{rcl} \{m,n & := & 0,0\} \\ (m = n \wedge x = 0,\ m = n \wedge x = 1) & \rightarrow & [m|\ m := m+1|m] \\ (m = n \wedge x = 1,\ m = n \wedge x = 0) & \rightarrow & [m|\ m := m+1|m] \\ (m = n+1,\ m = n+1) & \rightarrow & [m|\ n := m\,|n] \end{array}$$

in which a state invariant $n \leq m \leq n+1$ is clearly observed on all orbits.

Actually, a "wrong" value of n caused by a slow agent need not be the only source of errors that the designs like the ones we are considering can cope with. If, for instance, we have reasons to suspect that the adder employed in execution of implementation of actions may occasionally produce too small a value of the sum, we can offer the design:

$$\begin{array}{rcl} \{m,n & := & 0,0\} \\ (m = n \wedge x = 0,\ m = n \wedge x = 1) & \rightarrow & [n|\ m := n+1|m] \\ (m = n \wedge x = 1,\ m = n \wedge x = 0) & \rightarrow & [n|\ m := n+1|m] \\ (m > n,\ m > n) & \rightarrow & [m|\ n := m\,|n] \\ (m < n,\ m < n) & \rightarrow & [n|\ m := n\,|m] \end{array} \qquad (6)$$

in which we rely on the correctness of swap operation to cure possible errors of the adder.

While the implementation issues are not investigated in this report, a few remarks seem in order.

In our semantic model many things happen instantaneously: evaluation of a postguard, acceptance of an action results (*i.e.* the global state update), evaluation of all preguards and assignment of available agents to actions with satisfied preguards, all these happen in an "instant". Although other agents may continue their work in their private state-spaces, if they happen to finish their work in the same instant, each of them will be dealt with in a separate instant. How close to each other such instants are, is an implementation detail, and for a closed system it does not matter very much (under some obvious constraints of fairness). For an open system, however, a series of juxtaposed instants covers a finite interval, whose length also depends on an implementation. It is to be expected that with a large number of agents a long-lasting state will generate several copies of actions, whose starting times will be spread out, providing a uniformly dense cover of a certain portion of the state duration. It can be expected that such spread will offer a finer granularity of interactions with the outside world, to a certain extent compensating the slow speed of action executions by agents. Similarly, if we have reasons to believe that actual execution speeds of various agents are not identical, the instants of completion of copies of the same action undertaken in the same state will be scattered, and this scatter will provide a net for "catching" a particular postguard-satisfying state.

To get a feel for the actual behaviour of a system specified by (5), a simulation was run for up to a hundred agents ($N = 100$). All observations were made over 10 units of simulated time[6].

In Table 1, FFMTS stands for the flip-flop mean time to switch, *i.e.* the average time it takes to switch x from 0 to 1 or vice versa. When FFMTS is 0.3, the actual switches occur with uniform distribution within interval (0.2, 0.4), when FFMTS is 0.45, the corresponding interval is (0.30, 0.60).

OMC represents the observer mean cycle, *i.e.* the average time for execution of the guarded statement $m := m + 1$. When OMC is 0.15, the execution times are drawn uniformly from the interval (0.10, 0.20), when it is 0.075, the execution times cover the interval (0.050, 0.100).

[6]These data were first presented in [9].

FFMTS	0.3		0.3	0.45
OMC	0.15		0.075	0.15
N	AD	OMC/N	AD	AD
2	0.0500	0.0750	0.0249	0.0456
5	0.0303	0.0300	0.0167	0.0254
10	0.0145	0.0150	0.0064	0.0137
20	0.0074	0.0075	0.0034	0.0067
50	0.0029	0.0030	0.0014	0.0028
100	0.0016	0.0015	0.0008	0.0015

Table 1: Results of simulation

Finally, AD stands for the average delay in the detection of a flip-flop switch, *i.e.* for the observed average delay between the actual instant of the flip-flop switch and the instant at which the increased value of n was posted.

As expected, the average delays are inversely proportional to the number of agents and to their average speed. Thus, to obtain a more reliable (finer) observation one can simply increase the number of agents without necessarily making them any faster.

7 Another (not so simple) example

T. Nowicki in Example 3.1 of [4] presents the specification of a system whose exhibited behaviour strongly depends on the number of agents available for its implementation. Here we restate his specification in a form consistent with notation of this report and limited to a particular value of the parameter $N = 3$ (a "three-dimensional world"). By adding the initialization statement we avoid the need for much more general last element of Nowicki's specification.

Consider the specification

$$\begin{aligned} \{r,s,t &:= 0,0,0\} \\ (r=1 \wedge s=1 \wedge t=1, \mathbf{true}) &\rightarrow [r,s,t := 0,0,0] \\ (r=0, \mathbf{true}) &\rightarrow [r,s,t := 1,0,0] \\ (r=0, r=1) &\rightarrow [r,s,t := 1,1,0] \\ (r=0, r=1 \wedge s=1) &\rightarrow [r,s,t := 1,1,1] \end{aligned} \tag{7}$$

Let us interpret the three state coordinates as binary digits of a number that identifies (names) the state (thus, *e.g.* the state $r=1 \wedge s=1 \wedge t=1$ is identified by 7, while the state $r=1 \wedge s=0 \wedge t=0$ — by 1).

In a single agent implementation the only orbit of the system is {0,1}, all trajectories are finite and end in state 1.

In a two-agent implementation there are two possible orbits of (7): {0,1} and {0,1,3}, all trajectories are finite and end in states 1 or 3.

When the implementation uses three or more agents there are three possible orbits: {0,1}, {0,1,3} and {0,1,3,7}. While trajectories corresponding to the first two orbits are finite (and terminate in states 1 or 3), the last orbit includes infinite periodic trajectory 0,1,3,7,0,1,3,7,0,...Note that the longest orbit

includes also finite trajectories, in fact, any finite prefix of the periodic trajectory terminating in any state except 0 or 7 may be a trajectory of (7).

It is perhaps noteworthy that by observing the behaviour of the system specified by (7) one can infer the number of executing agents: if the system ever enters the state 7, the number of the agents must be at least three; if the system ever enters the state 3, the number of agents is at least two.

It is very interesting that the system specified by (7) can exhibit periodic behaviour only if at least three agents are assigned to its execution.

8 An intriguing relation to physics

Ever since the famous Bohr–Einstein disputes on essence of quantum physics, physicists seem to be fond of a class of mental experiments with an apparently paradoxical twist.

Briefly said, an experiment of this kind admits two perfectly well defined behaviours of observed objects. To be observed, each behaviour requires a specific arrangement of receptors that serve as the registering part of the experimental rig. The two arrangements are mutually exclusive: it is impossible to have them both in the same experiment simultaneously. (For example, a half-silvered mirror is, or is not, inserted at the cross point of two half-beams resulting from a previously split light beam.) The "metal" part of the experiment consists in verification that the behaviours detectable by specific receptors do occur.

The "mental" part of the experiment consists in reducing the initial conditions of the experiment so that also the *behaviours* become—apparently—mutually exclusive. (For example, if the light beam to be split is reduced to a single photon, after the split it may either travel—as a half-amplitude waves—along *both* paths, or—as a particle—along one of the two available paths.) Observations performed in thus reduced experiments are (assumed) consistent with patterns established by "metal" experiments. (For example, when the half-silvered mirror is in place, two half-amplitude waves restore a single photon wave by constructive interference, yielding evidence of travel accomplished along both paths; without the mirror the photon is registered by *one* of the two photodetectors aligned with two incoming paths, yielding evidence of travel accomplished along one path.)

The paradox part of the experiment consists in changing the receptor arrangement *after* the instant at which the possible behaviours became mutually exclusive. (For example, after the photon passed the first half-silvered mirror, the beam-splitter.) It appears that the selection of receptors influences the past selection of behaviour. (For example, placing, or not, of the second mirror "tells" the photon to *arrive* along both paths simultaneously, or along one path only.) Hence the paradox: cause succeeds effect[7].

In the behavioural specification style advocated in this report we can provide a perfectly matter-of-fact design for such experiments; *e. g.* for obvious reasons disregarding the input and output variables, we can write:

$$\begin{aligned} (\mathit{Photon\_available},\ \mathit{Second\_mirror\_in\_place}) &\rightarrow [\mathit{go\_by\_both\_paths}] \\ (\mathit{Photon\_available},\ \mathit{No\_second\_mirror}) &\rightarrow [\mathit{go\_by\_one\_path}] \end{aligned}$$

Both actions can be initiated in a state satisfying the predicate *Photon_available*. If there are more than one executing agents[8], both actions will be initiated and executed in parallel. Only one, however, can be accepted, depending on the situation existing in the state in which the action execution

[7] For a better exposition from a professional physicist *cf.* [10]

[8] The nature of executing agents is here somewhat unclear; for the moment let us just say that they constitute a part of a "mental experiment implementation".

terminates, the results of the other action are ignored (by design!) There is no paradox at all: having allowed (indeed, having insisted on!) maximal parallelism of actions—whatever may be started, is started as soon as possible—we see nothing strange in both virtual behaviours of the photon being attempted in parallel, and the actual behaviour being determined at the acceptance (*i.e. observation*) state by criteria *stated* in the design but *evaluated* at the instant of acceptance.

The ease with which a subtle (baffling?) physical behaviour can be described in the present style suggests several extensions:

- The notion of executing agents, quite familiar in computer-related research, may be too far-fetched in descriptions of behaviours of a more general kind. Perhaps it can be abstracted from, *e.g.* assuming availability of an infinite supply of agents. This should lead to a model in which all actions with satisfied preguards are continually started, as long as the system state satisfies the preguard. If the preguard is sufficiently weak, many different states will satisfy it, therefore the actions initiated under satisfaction of this preguard will have some sort of dependence on initial conditions in addition to the common characteristic of shared satisfied preguard.
- We can get rid of the auxiliary notion of executing agents by *postulating* that launching an action is an inherent property of a state which satisfies the action's preguard, just as it is an inherent property of a state to be susceptible to modification by a completed action if the state satisfies this action's postguard.
- In either case, various action distributions may be considered, reflecting possible preferences among actions whose preguards are satisfied.
- The state space may be considered as an *action field* without any dependence on any notion of global time.

We hope to investigate some of these model extensions in subsequent research.

9 Multiagent controller

The design method illustrated in Section 6 can be easily extended to on-line control problems.

Consider a simple version of such a problem. An aircraft flies in a plane on a course determined by its current position (x, y) and coordinates of the goal (X, Y). We assume that X and Y are fixed, and the instantaneous values of x and y are constantly available as values of the public variables x and y, respectively. Function $f.(x, y, X, Y)$ yields the setting of aircraft's controls consistent with the optimal trajectory between (x, y) and (X, Y). On board of the aircraft there is a radar that sweeps the plane in a more or less uniform circular scan. At each instant the value of public variable ϕ shows the current measure of the angle between aircraft's trajectory and the direction of the radar's beam. If an obstacle is detected, two public variables d and v_r provide values of measured distance to the obstacle and its radial (Doppler) velocity towards the aircraft.

Predicate $A.(\phi, d, v_r)$ is satisfied iff the obstacle presents a danger to the aircraft. Function $g.(\phi, d, v_r)$ yields the setting of aircraft controls compatible with the optimal avoidance manoeuvre. It is assumed that A is a total predicate, i.e. it is defined for all values of variables ϕ, d, v_r, including those that correspond to no obstacle detected (*e.g.* $d = \infty$) or an obstacle moving away from the aircraft ($v_r < 0$); similarly, g is assumed to be a total function, yielding some reasonable value for setting of controls when evaluated in a state where $\neg A$ holds. A simple controller may be specified as follows:

$$
\begin{array}{rl}
(\mathit{normal}, \mathit{normal}) \rightarrow & [x, y, \phi, d, v_r| \\
& \mathit{read}(x, y, \phi, d, v_r); \\
& \textbf{if } A.(\phi, d, v_r) \textbf{ then } \mathit{normal} := \textbf{false} \\
& \qquad\qquad \textbf{else } \mathit{controls} := f.(x, y, X, Y) \textbf{ fi} \\
& | \ \mathit{normal}, \mathit{controls}] \\
(\neg\mathit{normal}, \neg\mathit{normal}) \rightarrow & [x, y, \phi, d, v_r| \\
& \mathit{read}(x, y, \phi, d, v_r); \\
& \textbf{if } A.(\phi, d, v_r) \textbf{ then } \mathit{controls} := g.(\phi, d, v_r) \\
& \qquad\qquad \textbf{else } \mathit{controls} := f.(x, y, X, Y) \textbf{ fi}; \\
& \mathit{normal} := \textbf{true} \\
& | \ \mathit{normal}, \mathit{controls}] \\
(\neg\mathit{normal}, \mathit{normal}) \rightarrow & [x, y, \phi, d, v_r| \\
& \mathit{read}(x, y, \phi, d, v_r); \\
& \textbf{if } A.(\phi, d, v_r) \textbf{ then } \mathit{normal} := \textbf{false} \\
& \qquad\qquad \textbf{else } \mathit{controls} := f.(x, y, X, Y) \textbf{ fi} \\
& | \ \mathit{normal}, \mathit{controls}]
\end{array}
\tag{8}
$$

The read operations in the right-hand side actions have been included to underscore the fact that, although each processor assigned to an action gets a copy of the listed part of the state, corresponding variables may have to be polled (in the private copy of the state) and their values may have to be converted; in other words, it may take some time before the values needed for computations are available. action specified in (8) may be started in a normal situation (roughly: when no danger is perceived), its outcome is intended to be accepted also in a normal situation. This action, apart from reading the current position and radar data, consists of two mutually exclusive options: either a threat is detected and the public situation must be reclassified to abnormal, or a fresh setting of controls is to be worked out by evaluating f. may be started accepted in an abnormal situation, too. This action — when accepted — restores the normal classification of the public state either because the test indicates that the threat is there no more, and controls are set according to f algorithm, or because controls are set according to g algorithm.

Finally, the third action of (8) starts one. If the test confirms the presence of a threat, the public classification will be changed to abnormal. If the test fails (there is no imminent danger), controls are set according to f algorithm.

Note that the initialization of actions and acceptance of their results is governed by the value of *normal*, not by the results of evaluation of A; this illustrates an important principle of behaviour specifications: actions are selected according to the state classification; the classification procedure proper is a part of the actions.

Note also that the first action of (8), initiated in a state classified as normal, terminates as soon as this classification is negated by observations, without evaluating new settings for controls (it is assumed that if no value is assigned to an output variable no update happens and public variables retain their values). This reflects another design principle: abnormal (dangerous) situations need to be dealt with by actions designed for such situations; in this sense the similarity of actions in (8) is somewhat misleading.

In a more realistic treatment, the algorithms for control settings in the first and third actions may differ, *e.g.* the algorithm for f in the third action may take the advantage of the fact that it is meant to set optimal controls for ending the avoidance manoeuvre and getting back onto the normal course

(thus it may be different from the algorithm for f in the first action). If, however, as in the design (8), the right-hand sides of both actions are identical, we may use one of the transformation laws (*cf.* rule 2 of Section 5) and combine these two actions into one, with common right-hand side and postguard, and with preguard given by $normal \vee \neg normal = \mathbf{true}$, thus obtaining design

$$
\begin{array}{rl}
(\mathbf{true},\ normal) \rightarrow & [x, y, \phi, d, v_r| \\
& read(x, y, \phi, d, v_r); \\
& \mathbf{if}\ A.(\phi, d, v_r)\ \mathbf{then}\ normal := \mathbf{false} \\
& \qquad\qquad \mathbf{else}\ controls := f.(x, y, X, Y)\ \mathbf{fi} \\
& |\ normal,\ controls] \\
(\neg normal, \neg\ normal) \rightarrow & [x, y, \phi, d, v_r| \\
& read(x, y, \phi, d, v_r); \\
& \mathbf{if}\ A.(\phi, d, v_r)\ \mathbf{then}\ controls := g.(\phi, d, v_r) \\
& \qquad\qquad \mathbf{else}\ controls := f.(x, y, X, Y)\ \mathbf{fi}; \\
& normal := \mathbf{true} \\
& |\ normal,\ controls]
\end{array}
\tag{9}
$$

which neatly demonstrates two kinds of behaviour: one, permanently enabled (preguard **true**), setting controls for "normal" flight and signalling whenever a threat is detected, and another, engaged in while the situation is abnormal, setting the controls for an evasive manoeuvre.

As with the flip-flop-counter design, the multiagent implementation increases the chances for an early detection of a switch from dangerous to normal (and vice versa). Now, however, the prevention of the time-warp effects is a little more complex. We start again by considering the *controls* set in actions of (9) to be the "proposed" settings and by introducing *act_controls* for actual settings. Of course, we shall need fresh actions to execute the update of actual controls, (...,**true**) $\rightarrow$ [... *act_controls* := *controls* ...], but we lack a simple means of detecting obsolete values, such as was provided by the ratchet-like nature of the counter.

One way out of the difficulty would be to introduce a time-stamping mechanism and rely on its ratchet-like properties (control settings will be accepted as *act_controls* value only if they are fresher than the current value of *act_controls*).

There is, however, a different way, much more general, and directly related to the "physics" of the problem.

Denote a consistent[9] tuple (x, y, ϕ, d, v_r) by *observ*, hence $observ := read(x, y, \phi, d, v_r)$ is the action-internal operation representing the observation gathering. Let *hist* be a sequence of pairs $(observ, c)$, where c is of type **control_settings**. The maximal size of *hist*, its structure (list, queue, array, etc.) is left undecided in this paper. It is, however, assumed that:

1. there is available an operation $update.(hist, observ, c)$ which updates the *hist* variable by making the pair *observ*, c its "latest" element. (The undefinedness of *hist* size and structure allows us to ignore questions about deletions etc. contingent on execution of an update.)

2. there is available a test $compatible.(hist, observ)$ which yields **true** iff *observ* is a physically possible extension of *hist*. Some elements (conjuncts) of this test are obvious, for example, $(\phi_{observ} - last \cdot \phi_{hist} \geq 0) \bmod 2\pi$, others may depend on sophisticated extrapolation, sensitive to admitted flight manoeuvres.

[9] "Consistent" here means "relating to the same instant of observation".

3. both control setting functions f and g may advantageously use information contained in *hist*.

The controller specification becomes now:

$$
\begin{array}{rl}
(\textbf{true},\ normal) \rightarrow & [x, y, \phi, d, v_r, hist| \\
& observ := read(x, y, \phi, d, v_r); \\
& \textbf{if } A.(\phi, d, v_r) \textbf{ then } normal := \textbf{false} \\
& \quad \textbf{else } controls := f.(hist, x, y, X, Y) \\
& | \ normal,\ observ,\ controls] \\
(\neg normal, \neg\, normal) \rightarrow & [x, y, \phi, d, v_r, hist| \\
& observ := read(x, y, \phi, d, v_r); \\
& \textbf{if } A.(\phi, d, v_r) \textbf{ then } controls := g.(hist, \phi, d, v_r) \\
& \qquad\qquad \textbf{else } controls := f.(hist, x, y, X, Y); \\
& normal := \textbf{true} \\
& | \ normal,\ observ,\ controls] \\
(controls \neq act\_controls, \textbf{true}) \rightarrow & \\
& [observ, control,\ hist| \\
& \textbf{if } compatible.(hist,\ observ) \textbf{ then} \\
& \quad \textbf{begin} \\
& \quad act\_controls := controls; \\
& \quad update.(hist,\ observ,\ controls) \\
& \quad \textbf{end} \\
& | \ act\_controls,\ hist]
\end{array}
\tag{10}
$$

Further improvements to the design (10) are possible. The test upon which the setting of actual controls depends can be made sensitive not only to *hist* and *observ*, but also to the proposed controls, thus eliminating not only time-warp effects due to delayed processors but also a class of spuriously erroneous settings. Thus the design (10) has similar properties to those of (6).

In conclusion, it is perhaps worth observing that the described method of specifying control systems by doubly guarded actions of multiple processors totally avoids difficulties usually associated with integration of continuous and discrete components. Indeed, the continuous operation of the aircraft's radar (*i.e.* continuous changes of ϕ) and even the obviously continuous nature of its flight parameters (x, y) present no difficulty: corresponding variables are read only, *i.e.* instantaneous "snapshots" are made of their readings. This is quite apart from any mathematical constraints the continuity of the physical system imposes on the form of algorithms for functions f and g and on the form of condition *compatible*.

10 A classic example

Ever since Dijkstra published his elegantly stated problem of *Five Dining Philosophers* [1] it has become a standard case to study and apply novel cooperation (synchronization) techniques to. In this section we follow the established custom and assign a *dedicated agent* to each philosopher.

Traditionally, two problems are associated with Dining Philosophers:

- *deadlock* which arises when two or more philosophers enter a situation in which none can move on unless another one makes a move first; this is exemplified by the situation in which all

philosophers pick their left forks and wait each for her right neighbour to put the fork down so that they can pick it up (thus obtaining the set necessary for eating).

- *starvation* which is not a situation, but an infinite sequence of situations in which a philosopher is denied an action for which she is otherwise ready; this is exemplified by a conspiracy of two philosophers, who so coordinate their fork-pickings and -puttings that their common neighbour (sitting between the two) can never access both forks.

The following notation is localized to a chosen philosopher ("as seen by her"). The meaning of predicates is as follows

H	–	the philosopher is hungry
LOT	–	the left fork is on the table (free for taking)
ROT	–	the right fork is on the table
PL	–	the philosopher possesses the left fork
PR	–	the philosopher possesses the right fork

It is assumed that following actions are available to the philosopher (listed alongside their effects on predicates, if accepted)

take_l	makes PL **true** and LOT **false**
take_r	makes PR **true** and ROT **false**
release_l	makes LOT **true** and PL **false**
release_r	makes ROT **true** and PR **false**
think	has no effect on values of "fork" predicates
eat	has no effect on values of "fork" predicates

The philosopher's behaviour is quite naturally specified[10] by

$$
\begin{array}{lrcl}
 & \{LOT, ROT, H & := & \mathbf{true}, \mathbf{true}, \mathbf{false}\} \\
\text{(i)} & (\neg H, \mathbf{true}) & \rightarrow & [\mathit{think}; H := \mathbf{true}] \\
\text{(ii)} & (H \wedge PL \wedge PR, \mathbf{true}) & \rightarrow & [\mathit{eat}; H := \mathbf{false}; \mathit{release\_l}; \mathit{release\_r}] \\
\text{(iii)} & (H \wedge PL \wedge \neg R\_2, \neg ROT) & \rightarrow & [\mathit{release\_l}] \\
\text{(iv)} & (H \wedge PR \wedge \neg L\_2, \neg LOT) & \rightarrow & [\mathit{release\_r}] \\
\text{(v)} & (H \wedge LOT, LOT) & \rightarrow & [\mathit{take\_l}] \\
\text{(vi)} & (H \wedge ROT, ROT) & \rightarrow & [\mathit{take\_r}]
\end{array}
\tag{11}
$$

where $L\_2 \stackrel{\text{def}}{=} PL \vee LOT$ and $R\_2 \stackrel{\text{def}}{=} PR \vee ROT$.

Note that according to the adopted definitions, predicates PL and LOT (PR and ROT) cannot be both satisfied in any single state, but there could be states in which neither of the two holds (*e.g.* PL and LOT are both **false** when the left fork is held by the philospher's left neighbour). This reduces the total number of states that need to be considered for a complete specification of a philosopher's behaviour.

Line (i) specifies that whenever the philosopher is not hungry, she thinks and then becomes hungry; the always acceptable visible effect of this action is to set H to **true** in a finite time after a state satisfying $\neg H$ is established. Line (ii) specifies that whenever the philosopher is hungry and holds both forks, she starts eating, becomes not-hungry, and then releases both forks. The order in which the

[10] The specification given here differs a little from that in [7].

forks are released in the action program is immaterial: both "appear" on the table instantaneously when the results of this action are (unconditionally) accepted. The visible effects of this action are: both forks on the table and the not-hungry philosopher. Thus—in the world as seen by the philosopher—action (ii) is immediately followed by action (i); it is while the philosopher thinks that both forks are certainly available to her neighbours.

When the philosopher holds one fork only and the other fork is not on the table, she releases the fork she holds (line (iii) or (iv)); the result of this action—the released fork on the table—is accepted only if meanwhile the other fork is not put on the table (by a neighbour). Actions specified in these two lines prevent the exemplary deadlock.

Lines (v) and (vi) specify that the philosopher picks an available fork if she can do so before the fork disappears (*cf.* the postguard).

The only situation for which no action is specified is described by $H \wedge \neg(L\_2 \vee R\_2)$: the philosopher is hungry, holds no fork and no fork is on the table. This is not, fortunately, a stable state: the "missing" forks are held by neighbours, and they will eventually release them. When the philosopher is hungry and both forks are on the table, the choice with respect to which one to pick, *i.e.* the choice between lines (v) and (vi), is nondeterministic; as usual we expect it to be unbiased. Note that while both forks are released simultaneously (after eating), fork lifting is always sequential, specified by separate actions.

For all other situations well-defined actions are prescribed; if the results of these actions are not accepted at any time it is because a neighbour successfully completes an action of her own. In a system composed of several philosophers this means that some action is accepted in each state. Thus the specification (11) excludes not only the exemplary deadlock, but also any other; the design is deadlock-free.

It is perhaps worthwhile to observe that the specification (11) avoids deadlocks without referring to any information on the actual state of the philosopher's neighbours. This contrasts with the solution proposed by Dijkstra in [1], where the ith philosopher's behaviour is programmed with an explicit knowledge of the "state" (hungry/thinking/becoming hungry/eating) of philosophers $i-1$ and $i+1$. As soon as the information on neighbours is explicitly available and used, a conspiracy to starve a common neighbour becomes possible: to formulate a conspiratorial program a free access to such information is needed.

In our design, where this information is not available (nor needed), starvation is possible either because of a gross heterogeneity of philosophers, or as a chance phenomenon. It certainly cannot be excluded by the specification (11) that a very slow-acting philosopher is flanked by two nimble-fingered and fast-thinking ones. In that case the quick philosopher's entire postprandial activity may take less time than the middle philosopher's fork-lifting action. Such a pair of philosophers may in fact monopolize four forks (although each *must* put her forks on the table after each meal and leave them there for a *finite interval of time*). This illustrates the point about gross heterogeneity of philosophers. Whether to consider it as a deficiency of the design is a matter of taste: the practical consequences are not much different from classic solutions in which durations of philosophers eating times would be widely different.

Even if the philosophers are roughly similar, a pattern of fork movements preventing one or more of their number from eating would not be inconsistent with the design (11), just as an arbitrarily long run of heads is not inconsistent with repeated tossing of a fair coin (*cf.* [8]). This seems unavoidable, but practically irrelevant.

ACKNOWLEDGEMENTS. The research reported here was supported in parts by project CRIT-1 of the European Commission and USAF Special Contract F61708-94-C0001.

References

[1] E. W. Dijkstra. Hierarchical ordering of sequential processes. *Acta Informatica*, 1:115–138, 1971.

[2] E. W. Dijkstra. *A Discipline of Programming.* Prentice–Hall, 1976.

[3] J. La Salle and S. Lefschetz. *Stability of Liapunov's Direct Method.* Academic Press, 1961.

[4] T. Nowicki. A dynamical model of behavioural specifications. Technical Report TR-01(201), Institute of Informatics, Warsaw University, January 1995.

[5] B. Randell. System structure for software fault-tolerance. *IEEE Transactions on Software Engineering*, SE1:220–232, 1975.

[6] W. M. Turski. On programming by iteration. *IEEE Transactions on Software Engineering*, SE10(2):175–178, 1984.

[7] W. M. Turski. On specification of multiprocessor computing. *Acta Informatica*, 27:685–696, 1990.

[8] W. M. Turski. On starvation and some related issues. *Information Processing Letters*, 37:171–174, 1991.

[9] W. M. Turski. On doubly guarded multiprocessor control system design. In *Proceedings of the 4th DCCA Conference*, pages 1–11, San Diego, California, 1994. to appear in Springer-Veralag.

[10] J. A. Wheeler. The computer and the universe. *International Journal of Theoretical Physics*, 21:557–572, 1982.

A Appendix

A system of Five Dining Philosophers specified in Section 10 has been implemented by direct simulation of specification (11). To preserve the nondeterminism present when for a hungry philosopher both forks are on the table, a random choice was programmed between the two actions (v) and (vi).

I

N	E	ET	TT	MT
0	124	31.012	62.275	6.754
1	124	31.077	62.471	6.492
2	125	31.281	63.106	5.653
3	126	31.530	62.869	5.641
4	125	31.242	62.997	5.801

II

E	ET	TT	MT
159	39.765	40.833	19.410
157	39.305	40.103	20.599
160	39.936	40.567	19.505
162	40.496	41.291	18.220
158	39.600	41.176	19.232

III

N	E	ET	TT	MT
0	77	38.554	20.154	41.301
1	79	39.513	20.999	39.497
2	81	40.355	21.773	40.154
3	77	38.604	21.251	40.154
4	80	40.164	20.960	38.885

IV

N	E	ET	TT	MT
0	95	23.698	48.945	27.359
1	85	42.336	42.884	14.781
2	86	43.068	43.546	13.388
3	95	23.657	50.994	25.352
4	92	23.016	46.741	30.244

V

E	ET	TT	MT
97	24.251	48.967	26.898
94	46.848	47.017	6.251
101	25.300	51.692	23.124
93	46.523	46.640	6.593
95	23.707	49.547	26.862

VI

N	E	ET	TT	MT
0	114	28.532	57.598	13.872
1	116	29.102	58.277	12.622
2	112	28.064	56.256	15.680
3	115	28.738	57.351	13.912
4	114	28.429	57.351	13.912

VII

E	ET	TT	MT
106	26.592	52.792	20.736
107	26.690	53.438	19.992
90	22.537	45.119	32.436
98	24.564	48.980	26.576
101	25.256	51.063	23.800

VIII

N	E	ET	TT	MT
0	109	27.146	55.036	17.820
1	104	26.013	52.546	21.443
2	97	24.350	49.029	26.623
3	111	27.782	55.411	16.810
4	111	27.760	55.414	16.828

Each of the eight tables above presents statistics collected over a period of approximately 100 units of simulated time for various settings of experimental constants. In column N, common to each row of tables, the philosophers' ID numbers (0–4) are given. Column headed E records the number of meals consumed by each philosopher. In column ET the total "eating time" (spent in action (ii)) for each philosopher is given; similarly, column TT presents the total "thinking time" (spent in action (i)). Column MT—"management time"—records the difference between the (simulated) total time of a given experiment and the sum of values in columns ET and TT.

In all experiments the average durations of various actions were given in a table $T[i,j], i = 0,..,4,\ j = 0,..,5$. When the action j was to be executed by the ith philosopher the value of variable *totaltime* was increased by $(0.95 + 0.1 * random) * T[i,j]$ with *random* drawing uniformly distributed values from $[0,1)$, until accumulated *totaltime* exceeded 100.

For all experiments table T entries for the fork-releasing actions, (iii) and (iv) in specification (11), were set to 0.01.

Experiments I–III present system behaviour for various eating/thinking times ratios. In the experiment reported in Table I the average thinking time was set to 0.5, average eating time to 0.25. In the experiment of Table II both averages were set to 0.25, while in the experiment of Table III the average thinking time was set to 0.25, average eating time to 0.5. In all three experiments the average fork-picking durations (actions (v) and (vi)) were fixed at 0.01.

Tables IV and V present the influence of slow-eating philosophers on the system behaviour (and on their neighbours). In both experiments all thinking time averages were set identically to 0.5; fork-picking and -releasing durations were as in the first batch of experiments. In each experiment of this group two philosophers had their average eating time set to 0.5, while the same constants for remaining three philosophers were 0.25. In experiment IV the slow-eating philosophers were seated at places 1 and 2, in experiment V at places 1 and 3.

Finally, **Tables VI–VIII** present experiments designed to identify the influence of the fork-picking time constant on the system behaviour. In all experiments of the third row all settings are as in experiment I, except for the average duration of fork-picking by the philosopher seated at place 2. Table VI yields statistics on the behaviour obtained with the left-fork-picking constant of 0.1, Table VII of 0.2, while Table VIII presents the data obtained with the right-fork-picking constant of 0.2.

The aim of this Appendix is to illustrate a use one can make of specifications like (11) in analyzing various aspects of systems specified.